Algorithms for structural and dynamical polychronous groups detection

Régis Martinez[1]
Hélène Paugam-Moisy[2]
[1] LIRIS, UMR CNRS 5205, Université de Lyon, F-69676 Bron, France
[2] TAO - INRIA, LRI - Université Paris-Sud 11, F-91405 Orsay, France

October 21, 2009

Abstract

Polychronization has been proposed as a possible way to investigate the notion of cell assemblies and to understand their role as memory supports for information coding. In a spiking neuron network, polychronous groups (PGs) are small subsets of neurons that can be activated in a chain reaction according to a specific time-locked pattern. PGs can be detected in a neural network with known connection delays and visualized on a spike raster plot. In this paper, we specify the definition of PGs, making a distinction between structural and dynamical polychronous groups. We propose two algortihms to scan for structural PGs supported by a given network topology, one based on the distribution of connection delays and the other taking into account the synaptic weight values. At last, we propose a third algorithm to scan for the PGs that are actually activated in the network dynamics during a given time window.

1 Introduction

One of the main challenges in cognitive science is to understand how knowledge is represented and processed in the brain. From the early notions of cell assemblies [5] and "grand-mother cells" (see [4]), many open questions are still debated. What is the support of memory? How and where information is coded in the brain activity ? The recent hypothesis that information could be encoded by precise spike timings gives arguments for the thesis of temporal rather than spatial cell assemblies. Pointing out the fact that connection delays have non uniform values between neurons in the brain, Izhikevich proposed the concept of polychronization [7], which is far richer than the current concepts of synchronization and synfire chains [1]. Also in computer science the concept of polychronization yields valuable tracks for defining new learning rules in spiking neuron networks [10] and polychronous groups have been confirmed to play the role of dynamical cell assemblies in a classification task [9]. Studying PGs and

understanding their role in information coding could help improving both the network structure and the effectiveness of learning rules acting on delays.

So far, no formal definition has been given for a polychronous group, and only specific methods to inventory them have been proposed [6, 8]. In section 2 we give a precise definition of a polychronous group (PG), making a distinction between structural and dynamical PGs. In section 3 we present three algorithms and data structures to inventory all polychronous groups of a given network topology and to detect which groups are triggered in spike activity. In section 4 we give a complexity analysis of the algorithms, and we present experimental measurements: Number of PGs (mean values, from several experiments) when varying different parameters.

2 Definition of polychronous groups

2.1 PG definition

In the founding paper [7] polychronization denotes the fact that several neurons can be activated in a chain reaction according to a specific time-locked pattern of firings, not only in pure synchrony. A polychronous group is characterized by a precise temporal pattern of firings, in a subset of neurons, that is more likely to happen than just by chance in neural activity. Such patterns are intrinsically based on the network topology: connectivity, synaptic weights and especially conduction delays.

In recent work [8] a polychronous group, referred to as a *polygroup*[1], is defined as the set of neurons that supports the time-locked pattern. We state that the set of neurons involved in the temporal pattern is not enough to characterize the PG. Indeed, if one neuron can appear in more than one PG, it is clear that a given set of neurons could also fire with several different timings, and support or participate to more than one PG. The chain reaction that defines a PG is a series of causal interactions between neurons,

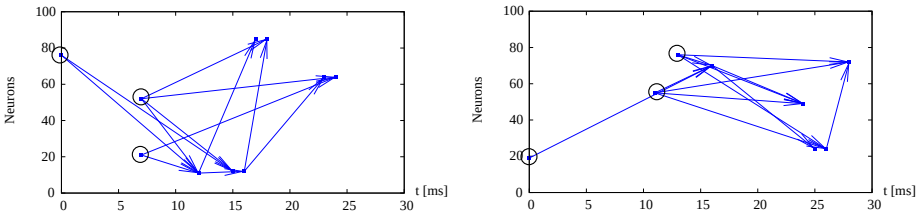

Figure 1: Graphs for two polychronous groups: the $21-52-76$ $(7,7,0)$ (left PG) and the $19-55-76$ $(0,11,13)$ (right PG). They share neuron 76 among their respective sets of triggers.

such as $[N_1,t_1],[N_2,t_2],...,[N_i,t_i] \Rightarrow [N_j,t_j]$, where a pair $[N_k,t_k]$ denotes a spike fired by neuron N_k at time t_k. The chain is started by a specific firing pattern of a small number s of neurons, the triggering neurons, further named *triggers*. Hence we propose to define a PG by a list of triggers associated to a temporal firing pattern:

[1] we do not use the term *polygroup* because it might bring confusion with other uses in Physics

Definition 1 *A s-triggered* **polychronous group** *refers to the set of neurons that can be activated by a chain reaction whenever the triggers $N_k (1 \leq k \leq s)$ fire according to the timing pattern $t_k (1 \leq k \leq s)$. The PG is denoted by:*

$$N_1 - N_2 - ... - N_s(t_1, ..., t_s)$$

where the firing times t_k are listed in the same order as the corresponding triggers N_k.

To decide whether a neuron could be activated by counting the number of spikes it recieves simultaneoulsy, we follow Izhikevich [7] by extending the notion of simultaneous arrival to a short time range denoted $jitter$.

A graphical representation of a PG can be plotted on a spike raster plot of the network activity, focusing attention on a subset of neurons, and drawing additional links for representing the causality of interactions (see Figure 1). Such a representation is a subgraph of the neural activity, where a vertex is a pair $[N_k, t_k]$, and a directed edge denotes the causal influence from a pre-synaptic neuron to a post-synaptic one.

2.2 Structural PGs vs Dynamical PGs

Definition 1 allows to inventory all the possible s-triggered polychronous groups supported by a given spiking neuron network with known connectivity and conduction delays, disregarding weight values. Since such PGs depend on the network architecture only, they are structural and we call them the **supported PGs**.

However, synaptic weights are usually subject to learning rules and their values change through time, which can influence the decision whether a certain amount of spikes simultaneoulsy incoming to a neuron N_j is sufficient or not for triggering a spike fired by N_j. Then we define **adapted PGs** by applying Definition 1 in taking into account the membrane potential dynamics of neurons and the values of synaptic weights for finding the causal relations yielding a neuron N_j to actually fire a spike at time t_j. Since activated PGs do not depend on the network dynamics (e.g. under the influence of a given input), they are also structural PGs.

Another question is to find which PGs actually appear in the spike activity of a neuron network during a given time window. These PGs are a subset of the structural PGs and we call them **activated PGs**. Unlike structural PGs, activated polychronous groups are dynamical PGs, since they depend on the network dynamics.

3 Scanning for polychronous groups

Algorithms 1 and 2 are designed to scan for supported PGs and adapted PGs respectively. They are based on a given network topology, with known connectivity, conduction delays and synaptic weights. In Algorithm 1, all the combinations of s neurons are tested as possible triggers, under the hypothesis that $NbSpikesNeeded$ are required to generate a causal relation making a neuron N_j to spike (s and $NbSpikesNeeded$ are set by the user). Actual post-synaptic potentials (PSPs) are not taken into account in Algorithm 1 whereas in Algorithm 2, the decision to let a neuron N_j spike requires

the computation of the weigthed PSPs recieved by N_j. In Algorithm 2, the amount of incoming spikes can differ from $NbSpikesNeeded$, a parameter which is no longer useful (s is still required and set by the user). Algorithm 3 is written to scan for the actual appearance of previously inventoried adapted polychronous groups in the network activity during a time slice of simulation data (with varying inputs, for instance).

All three algorithms are suited for any model of neuron that can be run in event-driven mode. Algorithm 2 uses the neuron model equation to decide whether the neuron spikes or not, on the basis of its recent PSP history. This constraint may be relaxed for Algorithm 1 that can be straightforward adapted to more general neuron models, since it is only based on spike events, and do not take care of the intrinsic dynamics of neurons.

The model neuron we used for experiments has the following characteristics (based on SRM_0 [2]): The firing threshold ϑ is set to a fixed negative value (e.g. $\vartheta = -50\ mV$) and the threshold kernel simulates an absolute refractory period τ_{abs}, when the neuron cannot fire again, followed by a reset to the resting potential u_{rest}, lower than ϑ (e.g. $u_{rest} = -65\ mV$). We assume that the simulation is computed in discrete time (with $0.1\ ms$ time steps). The variables of each neuron are updated at each new incoming spike (event-driven programming), which is sufficient for computational purpose.

The network structure is typical of Reservoir Computing methods: Connections between neurons are drawn randomly, according to a given connectivity; Weights start from 0.5 as initial value and can vary under STDP, a temporal Hebbian rule of synaptic plasticity; Delays are fixed but random, between 1 and 20 ms.

3.1 Definitions

3.1.1 Notations

n : number of neurons in the network
N_i : neuron numbered i
N_{T_k} : triggering neuron numbered T_k, with k from 1 to s
t : current time (virtual biological time, in simulations)
d_{ij} : conduction delay on the connection from N_i to N_j
w_{ij} : synaptic weight from N_i to N_j, equals 0 if connection does not exist.

3.1.2 Data structures

Post-Synaptic Potentials. A PSP is denoted PSP_{t_{psp},N_l,N_m} : Post-Synaptic Potential evoked at time t_{psp} by a pre-synaptic neuron N_l on a post-synaptic neuron N_m.

Event queue. The queue $EventQueue$ is the structure for processing simulation events (evoked PSPs). It contains the events ordered by their chronological occurence in the future.

PSP list stored in neurons. For the purpose of our algorithms, we need to store, for each neuron, the list of the spikes it recieved during a given time course elapsed, the $Jitter$ (see Section 2.1). This list is called $PSPList_i$, for neuron N_i.

Data structure for a PG. When a PG is calculated, informations have to be stored :

- timings $t_{T_1}, ..., t_{T_s}$ of the triggers N_{T_1} to N_{T_s};
- for each PSP, PSP_{t_{psp},N_l,N_m}, the ID of N_m and its spike-firing time, the ID of N_l and the time of evocation of the PSP.

3.1.3 Variables

$NbTriggeringConnections_i$: number of connections received by N_i from triggers
$MaxPotential_i$: maximum membrane potential that neuron N_i might reach under the action of triggers
$tLastSpike_i$: time of most recent firing of neuron N_i, initialized to $-RefractoryPeriode$, so that previous spikes are already out of refractory periode

3.1.4 Parameters

s : number of triggers, fixed for every polychronous groups
$NbSpikesNeeded$: number of simultaneous impinging spikes necessary to trigger a new spike in any neuron
$Jitter$: time range for spikes to be considered as "simultaneous"
$NbSpikesMax$: maximum number of spikes in a polychronous group
$NbSpikeMin$: minimum numer of spikes in a polychronous group for it to be saved
$MaxTimeSpan$: maximum time span of the polychronous group
$MinTimeSpan$: minimun time span of the polychronous group
$PSPStrength$: amplitude of a Post Synaptic Potential (PSP)
$RestingPotential$: default membrane potential of a neuron when it has no input
$Threshold$: membrane potential value above which the neuron spikes
$RefractoryPeriode$: time after a spike, during which a neuron cannot fire again

3.2 Algorithms

3.2.1 Algorithm 1

In order to list the supported PGs, we first look at all the combinations of a given number s of neurons. We check each combination, looking for neurons that might be excited enough to fire in turn, because they recieve more than a certain amount of spikes, $NbSpikesNeeded$. If such neurons exist, then the combination becomes a set of triggers. We simulate the firing of the triggers with the right starting timing and record the propagation of the neural activity, until it dies or it reaches an upper limit $MaxTimeSpan$ set for the time span of a PG. Moreover, we limit the record to a given number of neurons $NbSpikesMax$ in order to truncate the possible cyclic PGs.

In this algorithm, the propagation of the activity is based on the number of spikes recieved by each neuron in a time window. For instance, the neurons may be parameterized so that they fire whenever they are impinged by at least three spikes within a millisecond. A full description, in pseudo-language, is given in Annex.

3.2.2 Algorithm 2

The principle of this algorithm is very similar to the previous one, except that the decision of firing or not is based on the level of the membrane potential, which depends

on (a) the weights of the incoming connections that will modulate the increase of the membrane potential and thus the probability to generate a new spike and (b) the elapsed time since the previous PSP.

As in Algorithm 1, we look at all the possible combinations of s triggers, except that neuron activity is calculated upon its membrane potential exceeding or not the firing threshold. In this algorithm, the propagation of the activity is based on the fact that the mambrane potential exceedes the threshold. See Annex.

3.2.3 Algorithm 3

Algorithm 3 is written to scan for the appearance of known polychronous groups (already detected by Algorithm 1 or 2) in the activity recorded from a simulation, during a given time range.

In order to detect the activation of a PG in a particular time window in the recorded activity of a simulation (Algorithm 3), it would be ideal to check if the whole group is activated. For sake of computational time, we only look for the firing of the triggers of the PG, with the good timing pattern within a precision of $Jitter$. We based this algorithmic simplification on the assumption that the activation of the triggers will activate the tail of the polychronous group with little change, which is likely if the known PG has been previously detected to be an adapted PG, but could fail in case of supported PGs.

1: // We look for the actual activation of known PGs in a temporal range $[Start; End]$. For each neuron N_i, the list of spikes fired by this neuron between times $Start$ and End is stored already (i.e. we assume that the spike raster has been recorded).

2: Let $Spike_{mi}$ be the m^{th} spike of the list of spikes fired by N_i, at time $t_{Spike_{mi}}$

3: **for all** PG triggered by $\{N_{T_1}, N_{T_2}, ..., N_{T_s}\}$ with timing $\{t_{T_1}, t_{T_2}, ..., t_{T_s}\}$ **do**
4: $\quad \forall Spike_{mT_1}$ at time $t_{Spike_{mT_1}}$
5: $\quad$ **if** $\forall k \in [T_2; T_s], \exists\, Spike_{mT_k}$ at time $t_{Spike_{mT_1}} + (t_{T_k} - t_{T_1})$ **then**
6: $\quad\quad$ Save activation of PGs at time $t_{Spike_{mT_1}} - t_{T_1}$
7: $\quad$ **end if**
8: **end for**

4 Complexity study and experiments

4.1 Computational complexity

First, the complexity estimation for **Algorithm 1** is developed below. The complexity of Algorithm 2 and Algorithm 3 will be discussed afterwards.

Let c be the connectivity of a network, i.e. the probability that one neuron projects a connection to another. Let $AC = (n-1) \times c$ be the average number of connections recieved by any neuron in the network. Remind that s is the number of triggers.

The number of combinations to parse is $C_n^s = \frac{n!}{s!(n-s)!}$. For each combination, we search for neurons that recieve concurrent connections from triggers and count such connections. This step is computed with complexity $O(n + s \times AC)$.

The probability that a neuron recieves $k = NbSpikesNeeded$ connections from other neurons, with $k \leq s$, is c^k. We watch which neurons have enough input connections to trigger a new spike. For each excited neuron, we initialise the calculation of the corresponding PG and enqueue the spike events, i.e. $s \times AC$ operations.

The `while` block is the most difficult to evaluate because of the various parameters it envolves. In worst case, all n neurons will recieve $NbSpikesNeeded$ spikes, and spike in turn. Then there would be $n \times k$ events to process, and $n \times AC$ new PSP to enqueue. Neurons would spike again as soon as they can, in regard to their refractory period. There should be at most $M/R \times n \times AC$ events to process in the whole calculation, where $M = MaxTimeSpan$ and $R = RefractoryPeriode$.

Hence, the overall complexity X_{algo1} of Algorithm 1, in worst case, is :

$$\underbrace{C_n^s}_{combin.} \times [\quad \underbrace{(n + s \times AC)}_{search\ triggers} + \underbrace{(s \times AC) \times (c^k)}_{nb\ triggers} \times (\underbrace{s \times AC}_{init.\ spikes} + \underbrace{M/R \times n \times AC)}_{worst\ case\ events} \quad]$$

Since $s \ll n$, the number of combinations C_n^s can be approximated by $\left(\frac{n}{s}\right)^s$ and $AC = (n-1) \times c$ replaced by $c \times n$, which yields an upper bound for X_{algo1}:

$$X_{algo1} \leq \frac{1 + s \times c}{s^s}\, n^{s+1} + \frac{c^{k+2}}{s^{s-2}}\, n^{s+2} + \frac{M}{R} \times \frac{c^{k+2}}{s^{s-1}}\, n^{s+3} \tag{1}$$

It results that the complexity X_{algo1} is of order $O(n^{s+3})$ in worst case. In practical cases, spiking neuron networks usually have a sparse connectivity. Fixing the order of magnitude of the connectivity c to $1/\sqrt{n}$ looks like a realistic estimation, both for artificial networks (e.g. $c = 0.1$ in a network of 100 neurons, when computing with spiking neuron networks) and biological networks (around 10^5 connections per neuron between 10^{11} neurons in the human brain). Replacing c by $n^{-1/2}$ in Equation (1) gives:

$$X_{algo1} \leq \frac{1}{s^s}\, n^{s+1} + \frac{1}{s^{s-1}}\, n^{s+1/2} + \frac{1}{s^{s+2}}\, n^s + \frac{M}{R} \times \frac{1}{s^{s-1}}\, n^{s+1} \text{ since } k \geq 2 \tag{2}$$

Finally, in practical cases, the complexity of Algorithm 1 is of order $O(n^{s+1})$.

The complexity X_{algo2} of **Algorithm 2** is similar to X_{algo1} because both algorithms have the same control structure. Running Algorithm 2 should be slightly more time consuming if the neuron model is complex.

The complexity X_{algo3} of **Algorithm 3** is of order $O(P * S/n)$, where P is the number of known polychronous groups (computed by Algorithm 1 or 2), and S the total number of spikes in the time slice chosen for scanning the network activity.

4.2 Experiments

A direct comparison with other algorithms is not straightforward since the Izhikevich's code available on [6] starts from cutting off the weights under an arbitrary value of 0.95, and does not exactly compute any of the PG categories we defined. The Maier & Miller's method [8] (further noted "MM algo") is close to Izhikevich's one: both are based on a $n \times t_{max}$ matrix of spike arrival counts, with the risk of consuming a huge amount of memory, since t_{max} is the maximum time to which the simulation is run.

n	c	Jitter	supported PGs	adapted PGs	MM algo.
100	0.1	1ms	13.6	13.2	13.6
100	0.2	1ms	1295	1308	1300
100	0.18	1ms / 0.4	697	702 / 72	732
200	0.09	1ms / 0.4	295	274 / 16	289
500	0.036	1ms / 0.4	103	112 / 6	107
200	0.09	1.2ms	431		434
200	0.09	1.0ms	295		289
200	0.09	0.7ms	176		167
200	0.09	0.5ms	79		67
200	0.09	0.1ms	0		0

Table 1: Experimental measurements.

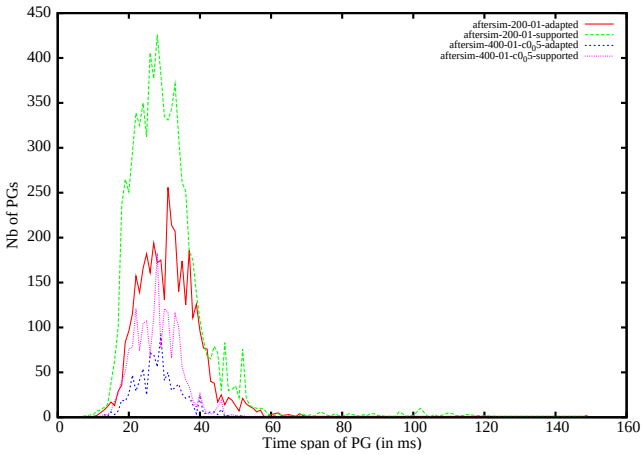

Figure 2: PGs timespan comparison.

In Table 1, $NbSpikesNeeded = s = 3$. The first two lines show the strong influence of the connectivity c on the number of PGs. In the next three lines, the network size is varied with an adapted value of c in order to keep fixed the degree of the connection graph. Different parameters of the neuron model (with coherent value of $Jitter$) highly influence the number of adapted PGs (but not sructured PGs). The last line shows that an absence of tolerance ($Jitter$ = $0.1ms$ = simulation time step) could lead to an absence of PGs. With $NbSpikesNeeded = 2$, $n = 100$ and $c = 0.18$ (not in Table 1), there are 387 supported PGs with $s = 3$ triggers, but only 4 with $s = 2$.

Figure 2 shows that the distribution of the PGs timespans (time from the spike of the earliest trigger to the spike of the latest neuron belonging to the PG) is similar for supported and adapted PGs, for different network sizes. Moreover, such a time span distribution is comparable to Izhikevich's observation ([7], p.127).

5 Discussion

We have proposed to clarify the definition of a polychronous group and set a standard notation, taking into account both the set of triggers and their specific firing pattern. We make a distinction between three categories of PGs, whether they are scanned from the network architecture only, or they take into account the variations of weights under a learning algorithm, or they are scanned for reflecting the dynamical activity inside the network in response to input data. Though designed for the analysis of simulations with spiking neuron network models, the algorithms could also be applied to real data since multi-neuron activities appear to be recordable in natural neuron networks [3].

References

[1] M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge Press, 1991.

[2] W. Gerstner and W. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[3] B. Gourévitch and J. Eggermont. Maximum decoding abilities of temporal patterns and synchronized firings. In *NeuroComp'2008*, 2008. hal-00331583.

[4] C.G. Gross. Genealogy of the "grandmother cell". *The Neuroscientist*, 2002.

[5] D.O. Hebb. *The Organization of Behaviour*. Wiley, New York, 1949.

[6] E.M. Izhikevich. http://vesicle.nsi.edu/users/izhikevich/publications/spnet.htm, 2006.

[7] E.M. Izhikevich. Polychronization: Computation with spikes. *Neural Computation*, 18(2):245–282, 2006.

[8] W.L. Maier and B.N. Miller. A minimal model for the study of polychronous groups. arXiv:0806.1070v1 [cond-mat.dis-nn], 2008. (presented at the TS4CF08 Meeting of The American Physical Society).

[9] R. Martinez and H. Paugam-Moisy. Les groupes polychrones pour capturer l'aspect spatio-temporel de la mémorisation. In *NeuroComp'2008*, 2008. hal-00331613.

[10] H. Paugam-Moisy, R. Martinez, and S. Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, 71(7-9):1143–1158, 2008.

A Algorithm 1

1: // Lines starting with // are comments.

2: **for all** combination of s neurons out of n neurons of the network **do**

// Look for PGs triggered by this combination

for all i from 1 to n **do**
  $NbTriggeringConnections_i = 0$
end for

// Count connections comming from triggers, to find common triggers output neurons
for all p from 1 to s **do**
  for all i with $w_{iT_p} \neq 0$ **do**
    $NbTriggeringConnections_i = NbTriggeringConnections_i + 1$
  end for
end for

for all p from 1 to s **do**
  for all i with $w_{iT_p} \neq 0$ **do**
    $NbPSP_i = 0$ // Reset count of PSP evoked in N_i in the last $Jitter$ ms

    if $NbTriggeringConnections_i \geqslant NbSpikesNeeded$ **then**
      // Reset $NbTriggeringConnections_i$
      $NbTriggeringConnections_i = 0$

      // A spike from N_i is triggered.
      We will calculate the PG with trigger neurons $\{N_{T_1}, N_{T_2}, \dots, N_{T_s}\}$ firing at N_i
      firing with timing $\{d_{max} - d_{T_1 i}, d_{max} - d_{T_2 i}, \dots, d_{max} - d_{T_s i}\}$
      with $d_{max} = max(d_{T_1 i}, d_{T_2 i}, \dots, d_{T_s i})$
      thus triggering neuron N_i.

      Add triggering spikes to PG. // Store triggering spikes to the PG data structure
      $PGSpikeCount = s$ // Count of spikes in this PG
      $t = 0$ // Initialise clock

      // Enqueue PSPs from triggers starting at $t = 0$.
      for all neuron N_h recieving a connection from $N_{T_k}, \forall k$ from 1 to s **do**
        Enqueue the new upcoming PSP evoked in N_h at time $t + (d_{max} - d_{T_k i}) + d_{T_k h}$ by the spike from N_{T_k}
      end for

      while $(PGSpikeCount < NbSpikesMax)$ and (PSP queue not empty) and $(t < MaxTimeSpan)$ **do**
        Consider next upcoming PSP PSP_{t_{psp}, N_l, N_m} evoked at time t_{psp} with
        N_l : firing pre-synaptic neuron of the spike that evoked PSP_{t_{psp}, N_l, N_m}
        N_m : post-synaptic neuron in which the PSP is evoked

        $t = t_{psp}$
        $NbPSP_m = NbPSP_m + 1$
        for all PSP_{t_{psp}, N_o, N_m} with $t - t_psp > Jitter$ **do**
          Erase PSP // Erase PSPs evoked in N_m older than $t - Jitter$ ms.
        end for

        if $(t - tLastSpike_m > RefractoryPeriode)$ and $(NbPSP_m \geqslant NbSpikesNeeded)$ **then**
          // N_m fires a spike
          $tLastSpike_m = t$
          Add a spike from N_m at time t, to PG
          $PGSpikeCount = PGSpikeCount + 1$
          for all neuron N_m recieving a connection from N_l **do**
            Enqueue an upcoming PSP evoked in N_m at time $t + d_{lm}$ by the spike from N_l
          end for
        end if
      end while

      if $PGSpikeCount > NbSpikeMin$ **then**
        Save the PG
      end if
    end if
  end for
end for

B Algorithm 2

N.B. Red printing is for the lines that differ from Algorithm 1.

for all combination of s neurons out of n neurons of the network **do**
  // Look for PGs triggered by this combination

  for all Neuron N_i, output of a triggering neuron **do**
    $NbPSP_i = 0$ // Count of PSP evoked in N_i in the last $Jitter$ ms
    $Potential_i = RestingPotential$ // Set initial membrane potential for N_i

    $MaxPotential_i = Potential_i + \sum_{T_j} w_{T_j i} PSPStrength$

    if $MaxPotential_i \geqslant Threshold$ **then**
      // A spike from N_i is triggered.
      We will calculate PG with trigger neurons $\{N_{T_1}, N_{T_2}, ..., N_{T_s}\}$ firing at N_i
      firing with timing $\{d_{max} - d_{T_1 i}, d_{max} - d_{T_2 i}, ..., d_{max} - d_{T_s i}\}$
      with $d_{max} = max(d_{T_1 i}, d_{T_2 i}, ..., d_{T_s i})$
      thus triggering neuron N_i.

      Add triggering spikes to PG. // Store triggering spikes to PG data structure
      $PGSpikeCount = 0$ // Count of spikes in this PG
      $t = 0$

      // Enqueue PSPs from triggers
      for all neuron N_h recieving a connection from $N_{T_k}, \forall k$ from 1 to s **do**
        Enqueue the new upcoming PSP evoked in N_h at time $t + (d_{max} - d_{T_k i}) + d_{T_k h}$ by the spike from N_{T_k}
      end for

      while $(PGSpikeCount < NbSpikesMax)$ and (PSP queue not empty) and $(t < MaxTimeSpan)$ **do**
        Consider next upcoming PSP PSP_{t_{psp}, N_l, N_m} evoked at time t_{psp} with
        N_l : firing pre-synaptic neuron of the spike that evoked PSP_{t_{psp}, N_l, N_m}
        N_m : post-synaptic neuron in which the PSP is evoked

        $t = t_{psp}$

        for all PSP_{t_{psp}, N_o, N_m} with $t - t_psp > Jitter$ **do**
          Erase PSP // Erase PSPs evoked in N_m older than $t - Jitter$ ms.
        end for

        // Re-evaluate decreasing membrane potential, with regard to last spike impact recieved t^f
        $Potential_m = \eta(Potential_l, t^f)$
        $Potential_m = Potential_m + w_{lm} \times PSPStrength$

        if $(Potential_m \geqslant Threshold)$ **then**
          // N_m fires a spike
          Add a spike to from N_m at time t, to PG
          $PGSpikeCount = PGSpikeCount + 1$
          for all neuron N_m recieving a connection from N_l **do**
            Enqueue an upcoming PSP evoked in N_m at time $t + d_{lm}$ by the spike from N_l
          end for
        end if
      end while

      if $PGSpikeCount > NbSpikeMin$ **then**
        Save the PG
      end if
    end if
  end for
end for